

UM ALGORITMO APROXIMATIVO APLICADO AO PROBLEMA DE PARTIÇÃO DE CONJUNTO

José Lassance de Castro Silva

Universidade Federal do Ceará, Departamento de Estat. e Matemática Aplicada
60000-000, Fortaleza-CE, e-mail: lassance@lia.ufc.br

Gerardo Valdísio Rodrigues Viana

Faculdade Lourenço Filho, Coordenação de Computação
60025-062, Fortaleza-CE, e-mail: valdisio@lia.ufc.br

Antonio Clécio Fontelles Thomaz

Universidade Estadual do Ceará, Departamento de Computação
60420-690, Fortaleza-CE, e-mail: clecio@lia.ufc.br

RESUMO

Este artigo apresenta um algoritmo aproximativo para resolver o Problema de Partição de Conjunto (*Set Partitioning Problem* - SPP). O SPP é um importante problema da otimização combinatória utilizado em muitas situações práticas. Transformamos o SPP num problema de otimização combinatória permutacional, e a resolução do problema é feita com base na avaliação de um número razoável de permutações que são analisadas através de um procedimento simples. Extensivos experimentos computacionais são descritos para instâncias do problema com até 55 linhas e 7479 colunas. O algoritmo obteve um bom desempenho quando comparado com os resultados encontrados na literatura.

Palavras-chave: Otimização Combinatória, Heurísticas, Permutação.

ABSTRACT

The aim of this paper is to present a technique for solving the Set Partitioning Problem (SPP). The SPP is an important problem used in many practical situations of the combinatorial optimization. We transform the SPP into a problem of permutational combinatorial optimization and your resolution is made with basis in the assign and evaluation of permutations. Extensive computational experiments are described for instances of the problem with up to 55 lines and 7479 columns. The algorithm got a good performance when comparative with the results found in literature.

Key- words: Combinatorial Optimization, Heuristics, Permutation.

1. INTRODUÇÃO

O problema da partição de conjunto, denotado na literatura por *Set Partitioning Problem* (SPP), é um problema de programação inteira/mista onde são dados:

- Uma matriz A , de ordem $m \times n$, com os $a_{ij} \in \{0, 1\}$, para todo $1 \leq i \leq m$ e $1 \leq j \leq n$;
- Um vetor custo C , com n elementos, onde cada c_j representa o custo associado da coluna j da matriz A .

O SPP consiste em determinar um subconjunto B de índices j das colunas da matriz A que cobre todas as suas linhas com o menor custo associado. Em outras palavras, o conjunto B é tal que: para cada $1 \leq i \leq m$ existe apenas um $a_{ik}=1$, com $k \in B$, e $\sum_{k \in B} c_k \leq \sum_{j \in B'} c_j$, para todo $B' \subset \{1, 2, \dots, n\}$.

O SPP pode ser formulado matematicamente com o seguinte modelo:

$$\text{Minimizar } z = \sum_{j=1}^n c_j x_j \quad (1)$$

$$\text{Sujeito a: } \sum_{j=1}^n a_{ij} x_j = 1, \forall i \in I = \{1, 2, \dots, m\} \quad (2)$$

$$x_j \in \{0, 1\}, \forall j \in J = \{1, 2, \dots, n\} \quad (3)$$

onde:

- $x_j = \begin{cases} 1, & \text{se o índice } j \in B \\ 0, & \text{caso contrário.} \end{cases}$
- A função objetivo (1) minimiza a soma dos custos associados as linhas cobertas de A ;
- O conjunto de restrição do tipo (2) determina o cobrimento de cada linha da matriz A ; e
- O conjunto de restrição do tipo (3) faz com que as variáveis do problema sejam binárias, de acordo com a definição das mesmas.

Alguns resultados podem ser observados para o SPP, através do modelo matemático acima, tais como:

- a) Se $m > n$ o SPP não apresenta solução viável para o problema;
- b) Existem subconjuntos de J que não representam uma solução viável para o problema;
- c) O SPP é NP-Completo com complexidade na ordem de $O(2^n)$;
- d) O modelo terá solução vazia se existir um $l \in I$ tal que $a_{lj}=0$, para todo $j \in J$.

Existem, na literatura, várias descrições para as aplicações do SPP, dentre as quais podemos citar [1, 2, 3, 4, 5, 6, 7]. Porém, o número de maior aplicação do SPP, que encontramos, foi relacionado com o problema de escalonamento de aeronaves para atender os vôos de determinadas linhas aéreas (*airline crew scheduling*), onde o objetivo principal é determinar a melhor coleção de rotas tal que cada vôo seja coberto por exatamente uma rota.

Devido a grande quantidade de aplicações do SPP, também encontramos um grande número de algoritmos que têm sido desenvolvidos para resolver o problema. Estes podem ser classificados em duas categorias: algoritmos exatos, que determinam a solução ótima do problema, e algoritmos aproximativos (heurísticas) que determinam boas soluções num tempo computacional razoável.

O ponto inicial para muitos algoritmos exatos é resolver o modelo matemático (1-3) de forma parcial ou total, através das técnicas de Relaxação Lagrangeana, Plano de Cortes com o Simplex, Branch and Bound, etc. cf. [3, 6, 8, 9]. Na resolução do problema utilizando heurísticas podemos citar [10, 11, 12, 13, 14, 15, 16, 17, 18], que fazem uso das mais variadas idéias para a resolução do SPP, dentre as quais aquelas descritas em Simulated Annealing, Algoritmos Genéticos, etc.

Este artigo trata especificamente do Problema de Partição de Conjuntos, nele apresentamos uma técnica para resolver o problema de forma aproximativa, denominada de Heurística Combinatória (HC). Esta técnica baseia-se na heurística permutacional desenvolvida por Silva e Soma [19]. Aplicamos HC aos problemas testes encontrados na literatura, e descrevemos os resultados dos nossos experimentos computacionais que comprovaram o bom desempenho da técnica.

A organização deste trabalho segue à: Na Seção 2, apresentaremos a heurística combinatória proposto, por nós, para resolver o problema SPP. Na Seção 3, fazemos a aplicação de HC aos

problemas gerados pela literatura. As conclusões estão apresentadas na seção 4, enquanto a bibliografia será apresentada na Seção 5.

2. A HEURÍSTICA COMBINATÓRIA

Um problema de otimização combinatória permutacional pode ser definido por um terno (S, g, m) , onde S é o conjunto de todas as soluções viáveis (soluções que satisfazem as restrições do problema, com $|S| = m!$), g é uma função ou um procedimento que avalia cada solução $s \in S$ do problema e m é uma instância do problema. O objetivo é encontrar a solução $s \in S$ que optimize a função objetivo do problema. Podemos representar s como uma permutação de m elementos distintos, ou seja, $s = \langle a_1 a_2 \dots a_m \rangle$. $N(s)$ é chamada a vizinhança de s e contém todas as soluções que podem ser alcançadas de s por um simples movimento. Aqui, o significado de um movimento é aquele de um operador que transforma uma solução para uma outra com pequenas modificações.

Silva e Soma [19] desenvolveram uma técnica para resolver POCP, denominada Heurística Permutacional (HP). Através desta técnica conseguimos intensificar o processo de diversificação, aumentando o número de soluções a ser avaliada no problema. O procedimento HP consiste basicamente em dividir o conjunto de soluções viáveis S em m vizinhanças $N(s_i)$ distintas entre si, e cada uma destas vizinhanças em quatro subvizinhanças $N(s_{ij}) \subset N(s_i)$, $1 \leq j \leq 4$. A permutação s_i , que gera $N(s_i)$, tem o elemento i na primeira posição da permutação, e para toda permutação s em $N(s_i)$, s também inicia com o elemento i . Desta forma $N(s_i) \cap N(s_k) = \emptyset$, $1 \leq i, k \leq m$, com $i \neq k$. Exceto $s_{i1} = s_i$, as quatro permutações s_{i1} , s_{i2} , s_{i3} e s_{i4} que produzirão as quatro subvizinhanças $N(s_{i1})$, $N(s_{i2})$, $N(s_{i3})$ e $N(s_{i4})$, respectivamente, são obtidas da troca de posições dos elementos de s_i , da seguinte forma:

- 1) A permutação s_{i2} mantém a primeira posição de s_i e inverte as $(m-1)$ posições restantes de s_i ;
- 2) A permutação s_{i3} mantém a primeira posição de s_i e troca sequencialmente 2 a 2 as demais posições adjacentes de s_i ;
- 3) A permutação s_{i4} mantém a primeira posição de s_{i3} e inverte as $(m-1)$ posições restantes de s_{i3} .

A subvizinhança $N(s_{ij})$ é formada por todas as permutações que são obtidas de s_{ij} trocando de posição 2 a 2 todos os elementos de s_{ij} , a partir da segunda posição, e mantendo a ordem dos outros elementos inalterado. Assim, o número de permutações em cada subvizinhança é $[(m-1) \times (m-2) / 2 + 1]$ que implica em $[4 \times (m-1) \times (m-2) / 2 + 1]$ permutações em cada vizinhança e um total de $[2 \times m \times (m-1) \times (m-2) + 4m]$ permutações geradas para o problema.

Dada uma permutação qualquer s_1 , podemos obter os demais s_i , $2 \leq i \leq m$, trocando somente o elemento da primeira posição de s_1 pelo elemento da i -ésima posição. Sem perdas de generalidades podemos supor $s_1 = \langle 1 2 3 \dots m \rangle$, então $s_2 = \langle 2 1 3 4 5 \dots m \rangle$, $s_3 = \langle 3 2 1 4 5 \dots m \rangle$, ..., $s_m = \langle m 2 3 4 5 \dots (m-1) 1 \rangle$.

Adotamos o seguinte critério para aumentar o número de soluções avaliadas no problema, com HP. Seja $s \in N(s_i)$, para algum i entre 1 e m , isto implica que $s = \langle i a_2 \dots a_m \rangle$. Sejam $v = \lfloor (m-1)/2 \rfloor + 2$, $p = v - \lfloor m/4 \rfloor - 1$ e $q = v + \lfloor m/4 \rfloor$. Tendo s a seguinte forma:

Solução		α	β	δ	μ
$s =$	$\langle i$	$a_2 a_3 \dots a_p$	$a_{p+1} \dots a_{v-1}$	$a_v a_{v+1} \dots a_{q-1}$	$a_q \dots a_{m-1} a_m \rangle$

Assim, a permutação s , a partir da segunda posição, é dividida em 4 subconjuntos α , β , δ e μ sendo que: se k é o resto da divisão de m por 4, $i.e.$, $k = m \text{ módulo } 4$, então temos que:

$$k = \begin{cases} 0 \Rightarrow |\alpha| = \lfloor m/4 \rfloor - 1 \text{ e } |\beta| = |\delta| = |\mu| = \lfloor m/4 \rfloor \\ 1 \Rightarrow |\alpha| = |\beta| = |\delta| = |\mu| = \lfloor m/4 \rfloor \\ 2 \Rightarrow |\mu| = \lfloor m/4 \rfloor + 1 \text{ e } |\beta| = |\delta| = |\alpha| = \lfloor m/4 \rfloor \\ 3 \Rightarrow |\alpha| = |\mu| = \lfloor m/4 \rfloor + 2 \text{ e } |\beta| = |\delta| = \lfloor m/4 \rfloor \end{cases}$$

Construa s^1, s^2, \dots, s^{23} a partir de s permutando os 4 conjuntos α, β, δ e μ . Assim $s^1 = \langle i \alpha \beta \mu \delta \rangle$, $s^2 = \langle i \alpha \mu \delta \beta \rangle$, ..., $s^{23} = \langle i \mu \delta \beta \alpha \rangle$.

Portanto, com este novo procedimento adicionado a HP, denominado HP*, o número total de permutações a serem avaliadas será de $[48 \times m \times (m-1) \times (m-2) + 96m]$, pois para cada permutação gerada em HP foram geradas mais 23 (vinte e três) permutações. Implementamos e executamos estes procedimentos recursivamente, sem analisar o desempenho da função objetivo do problema, e constatamos que não houve repetição de permutação para valores de $m \geq 10$. A Tabela 1, dada a seguir, apresenta o número de permutações geradas por HP e HP*.

<i>m</i>	HP	HP*	<i>m</i>	HP	HP*
10	1.480	35.520	70	657.160	15.771.840
15	5.520	132.480	75	810.600	19.454.400
20	13.760	330.240	80	986.240	23.669.760
30	48.840	1.172.160	85	1.185.580	28.453.920
40	118.720	2.849.280	90	1.410.120	33.842.880
50	235.400	5.649.600	100	1.940.800	46.579.200
60	410.880	9.861.120	200	15.761.600	378.278.400
65	524.420	12.586.080	500	248.504.000	5.964.096.000

Tabela 1 – Número de permutações avaliadas pelos procedimentos HP e HP*.

Algoritmo Guloso Simples – AGS

Input: m, n, A, C, I, J ;

P1) $W_j = \{i \in I; a_{ij} = 1\}$, para todo $j \in J$.

P2) $B = \emptyset$;

P3) $R_j = W_j$, para todo $j \in J$;

P4) While $(\bigcup_{j \in B} W_j \neq I)$ do

$$P4.1) k = \{l \in J; R_l \neq \emptyset \text{ e } \frac{c_l}{|R_l|} \leq \frac{c_j}{|R_j|}, \forall j \in J \text{ e } R_j \neq \emptyset\};$$

P4.2) $B = B \cup \{k\}$;

P4.3) $R_j = R_j - W_j$, para todo $j \in J$;

$$P5) g = \sum_{j \in B} c_j$$

Output: g e B .

Figura 1 – Pseudo-Código do algoritmo guloso simples.

O SPP pode ser definido como um terço $P=(S, g, m)$, de acordo com a definição de um problema de otimização combinatória permutacional, da seguinte forma:

- a) Um elemento s do conjunto de soluções S é representado por uma permutação das m linhas da matriz A , com a ordem de s determinando a seqüência na qual as linhas poderão ser cobertas;
- b) A função g que avalia uma solução de S é dada pelo procedimento HC, descrito a seguir na Figura 2. Porém, apresentamos na Figura 1, um algoritmo que determina uma solução para o problema, denominado Algoritmo Guloso Simples (AGS).

Heurística Combinatória - HC

Input: m, n, A, C, I, J, npg ; /* npg é o número de permutações geradas por HP* */

- 1) $W_j = \{i \in I; a_{ij}=1\}$, para todo $j \in J$.
- 2) $B = \emptyset$;
- 3) $Opt = +\infty$; /* Opt é um número bastante grande */
- 4) For $q=1$ until npg do
 - 4.1) $R_j = W_j$, para todo $j \in J$;
 - 4.2) $l=1$;
 - 4.3) $s=s_q$; /* s_q é a q -essima permutação gerada por HP* */
 - 4.4) While ($l \leq m$) do
 - 4.4.1) $i=s[l]$;
 - 4.4.2) If ($i \in (I - \bigcup_{j \in B} W_j)$) then
 - 4.4.2.1) $k = \{t \in J; R_t \neq \emptyset, i \in R_t \text{ e } \frac{c_t}{|R_t|} \leq \frac{c_j}{|R_j|}, \forall j \in J \text{ e } R_j \neq \emptyset\}$;
 - 4.4.2.2) $B=B \cup \{k\}$;
 - 4.4.2.3) $R_j = R_j - W_j$, para todo $j \in J$;
 - 4.4.3) $l=l+1$;
 - 4.5) $g = \sum_{j \in B} c_j$;
 - 4.6) If ($g < Opt$) then { $Opt=g; B^*=B; \}$

Output: Opt e B^* .

Figura 2 – Pseudo-Código da Heurística Combinatória.

Em HC, a ordem na qual as linhas da matriz A são cobertas é dada em função da permutação s , que foi gerada pelo procedimento HP*. A Heurística Combinatória gera um grande número de soluções viáveis (associadas às permutações) para o problema, e dentre elas seleciona aquela com o menor custo. Se fosse possível percorrer todas as permutações de 1, 2, ..., m , então a solução ótima seria encontrada para o problema. Vale ressaltar, que no procedimento HC, supõe-se que o modelo matemático do SPP não apresenta solução vazia.

3. EXPERIMENTOS COMPUTACIONAIS

Um grande número de experimentos computacionais foram realizados para observarmos o desempenho do nosso algoritmo, entretanto selecionamos aleatoriamente apenas 20 problemas para serem apresentados aqui. HC foi executado em um AMD-Athlon (1,3 GHz com 256 Mb de RAM) e o código

foi implementado em Fortran PowerStation 4.0. As classes de testes utilizadas foram definidas na biblioteca OR-Library [20], e a amostra considerada tem tamanho variado para m e n .

A Tabela 2, dada a seguir, apresenta os resultados obtidos para as diversas instâncias utilizadas nos nossos experimentos computacionais. Nessa tabela encontram-se: os valores de n , m e da solução ótima (*ótimo*) das instâncias; os valores de z na solução ótima, o GAP ($z/\text{ótimo}$) e o tempo gasto (em segundos) para AGS e HC; e a descrição da referência do problema na OR-Library.

Instância			AGS			HC			Referência
n	m	Ótimo	z	GAP	Tempo	z	GAP	Tempo	OR-Lib.
18	1072	8904	9224	1,036	0,01s	8904	1,000	1m00s	NW43
18	1210	8298	8298	1,000	0,01s	8298	1,000	1m47s	NW28
18	2540	4274	5774	1,351	0,06s	4547	1,063	2m03s	NW29
19	294	14877	15600	1,048	0,02s	14877	1,000	58s	NW32
19	404	10809	11526	1,066	0,03s	11343	1,049	7,25s	NW40
19	711	12534	n/f	–	0,01s	12806	1,021	1m31s	NW23
19	1366	6314	6780	1,074	0,07s	6568	1,040	2m21s	NW24
20	899	10488	10488	1,000	0,03s	10488	1,000	2m02s	NW34
20	1217	5960	6478	1,087	0,01s	6246	1,048	2m05s	NW25
22	685	16812	18645	1,109	0,02s	17772	1,057	2m03s	NW20
22	1355	9933	15969	1,607	0,05s	10752	1,082	3m29s	NW27
23	619	6984	7192	1,030	0,02s	7208	1,032	2m01s	NW22
24	434	35894	44636	1,244	0,05s	37294	1,039	2m17s	NW10
25	677	10080	11793	1,170	0,02s	10545	1,046	2m27s	NW39
27	626	14118	16394	1,161	0,02s	14190	1,005	2m53s	NW12
31	467	67743	67749	1,000	0,02s	67743	1,000	2m57s	NW15
36	5172	5476	5770	1,053	0,17s	5770	1,053	5m13s	NW07
40	3103	44108	46912	1,064	0,27s	44194	1,002	18m55s	NW09
40	2879	10898	12916	1,185	0,16s	11956	1,097	13m42s	NW19
55	7479	1086	1116	1,027	1,19s	1086	1,000	27m23s	KL01

Tabela 2 – Resultados dos experimentos computacionais.

A Tabela 3, dada a seguir descreve de forma sucinta as comparações entre AGS e HC, com base nos dados apresentados da Tabela 2.

Aspecto	AGS	HC
Determinação da Solução Ótima	Encontrados em apenas 3 problemas (NW28, NW34 e NW15).	Encontrados em 6 problemas (NW28, NW34, NW15, NW43, NW32 e KL01), e mais 2 problemas (NW09 e NW12) o GAP foi inferior a 1,01.
GAP superior a 1,05	Encontrados em 12 problemas.	Encontrados em 5 problemas.
Pior desempenho	Obtido no problema NW27 com GAP de 1,607.	Obtido no problema NW19 com GAP de 1,097.
Não apresentou solução viável	Apenas no problema NW23.	Apresentou solução viável para todas as instâncias dos problemas experimentados.
Média dos GAPs	1,122	1,031
Desvio Padrão	0,145	0,038

Tabela 3 – Comparação dos resultados, obtidos experimentalmente, de AGS com HC.

Referente ao tempo de execução de HC e AGS, verifica-se que AGS é muito rápido, tendo em vista que ele é um procedimento on-line em m , enquanto HC é polinomial. Entretanto, HC gerou soluções de boa qualidade, com um tempo computacional aceitável, para o tamanho das instâncias experimentadas.

4. CONCLUSÕES

Atualmente não é fácil desenvolver algoritmos que usem *processos de diversificação* e *intensificação* no espaço de soluções viáveis. No *processo de diversificação*, o objetivo é direcionar a busca para novas regiões, de forma a atingir todo o espaço de soluções possíveis. Enquanto que no *processo de intensificação* há um reforço à busca na região (vizinhança) de uma solução historicamente considerada boa. O algoritmo aqui proposto trabalha plenamente com o processo de diversificação para qualquer problema de otimização combinatória permutacional (POCP), enquanto que o processo de intensificação depende da característica de cada problema. O processo de diversificação não se faz presente na maioria dos algoritmos desenvolvidos para os POCPs devido os mesmos trabalharem com geradores de soluções aleatórias. De acordo com os nossos experimentos, gerar soluções aleatórias que envolvem permutação tem um custo muito alto, pois não é simples conter a repetição das permutações geradas, e uma das conseqüências disso é concentrar as buscas em um número pequeno de regiões que não atinge todo o espaço de soluções viáveis do problema.

Nossos experimentos computacionais baseados em problemas da literatura, indicam que é possível encontrarmos boas soluções quando utilizamos procedimentos construtivos que diversificam a busca dentro do conjunto de soluções viáveis. HC foi aplicado com sucesso no SPP, e acreditamos que não seria diferente se aplicássemos também a outras classes de problemas de otimização combinatória, que envolve permutação.

Uma forma de diminuirmos a complexidade de HC seria usar o processamento paralelo ou distribuído, com o intuito de dividir em tarefas, as gerações e avaliações das permutações configuradas por HP^* , a serem processadas pelos processadores existentes e disponíveis.

Estamos desenvolvendo outros procedimentos para aumentarmos o número de soluções a serem avaliadas por HP^* , com o intuito de dar mais abrangência a busca, sem que haja a perda no processo para o controle das soluções não repetidas.

Agradecimentos

Os autores agradecem apoio financeiro do CNPq, da Faculdade Lourenço Filho, da Universidade Federal do Ceará e da Universidade Estadual do Ceará.

5. BIBLIOGRAFIA

- [01] P. C. Chu and J. E. Beasley. A Genetic Algorithm for the Set Partitioning Problem. *Technical Report of the Management School Imperial College*, 1995.
- [02] J. Arabeyre, J. Fearnley, F. Steiger and W. Teather. The airline crew scheduling problem: a survey. *Transportation Science*, 3(2): 140-163, 1969.
- [03] E. Balas and M.W. Padberg. Set Partitioning: A Survey. *SIAM Review*, 18, 710-760, 1976.
- [04] J. Barutt and T. Hull. Airline Crew Scheduling: supercomputers and algorithms. *SIAM News*, 23(6), 1990.
- [05] I. Gershkoff. Optimizing flight crew schedules. *Interfaces*, 19:29-43, 1989.
- [06] K. Hoffman and M.W. Padberg. Solving Airline Crew Scheduling Problems by Branch and Cut. *Management Science*, 39, 657-682, 1993.
- [07] R. Marsten and F. Shepardson. Exact Solution of crew scheduling problems using the set partitioning model: recent successful applications. *Networks*, 11, 1981.
- [08] M. Fisher and P. Kedia. Optimal solution of set covering/partitioning problem using dual heuristics. *Management Science*, 36(6):674-688, 1990.
- [09] F. Harshe and G. I. Thompson. The column subtraction algorithm: an exact method for solving weighted set covering, packing and partitioning problems. *Computers & Operations Research*, 21(6): 689-705, 1994.
- [10] D. M. Ryan and J. C. Falkner. On the integer properties of scheduling set partitioning models. *European Journal of operational Resesarch*, 35: 422-456, 1988.
- [11] D. Levine. *A parallel genetic algorithm for the set partitioning problem*. PhD thesis, Illinois Institute of Technology, Department of Computer Science, May/1994.
- [12] P. C. Chu and J. E. Beasley. Constraint Handling in Genetic Algorithms: The Set Partitioning Problem. *Journal of Heuristics*, 11: p.323-357, 1998.
- [13] M. Fisher and L. Wolsey. On the Greedy Heuristic for Covering and Packing Problems. *SIAM Journal on Algebraic and Discrete Methods*, 3, 584-591, 1982.
- [14] N. Hall and D. Hochbaum. A fast approximation algorithm for the multicovering problem. *Discrete Applied Mathematics*, 15:35-40, 1989.
- [15] A. Feo and G. C. Mauricio and A. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, 8, 67-71, 1989.
- [16] C. R. Reeves, editor. *Modern Heuristics Techniques for Combinatorial Problems*. Blackwell Scientific, 1993.

- [17] M. Aourid and B. Kaminska. Neural Networks for the Set Covering Problem: An Application to the Test Vector Compaction. *IEEE International Conference on Neural Networks Conference Proceedings*, 7, 4645-4649, 1994.
- [18] E. Balas and M.C. Carrera. A Dynamic Subgradient-based Branch-and-Bound Procedure for Set Covering. *Operations Research*, 44, 875-890, 1996.
- [19] J. L. C. Silva e N. Y. Soma. Uma Heurística para Problemas de Otimização Combinatória Permutacional. *Proceedings of the XXXIII SBPO*, Campos do Jordão-SP, Brazil, 2001.
- [20] J. E. Beasley. OR-Library: Distributing Test Problems by Eletronic Mail. *Journal of the Operations Research Society*, 41: 1069-1072, 1990.